INTEL CORPORATION
5200 N.E. Elam Young Parkway
Hillsboro, Oregon 97124-9987
(503) 681-8080

Dear Customer:

Thank you for purchasing Intel's Motif* for Intel's X11 Release 3.

Intel is very pleased to be providing you with a complete Motif environment including the "uil" compiler.

The Product that you have received contains the following:

- Motif Software five 5-1/4" diskettes and five 3-1/2" diskettes
- Motif User's Guide
- Release Notes (includes mwm documentation)

If any of these items are missing, please call the appropriate number from the list below.

Installation support for Intel Motif software is included in the one hour of installation support that you received with your purchase of Intel's UNIX System V, Release 4.0. For more extensive support agreements, contact the Intel support organization at the phone number listed below.

To receive phone installation support, please call the appropriate number from the following list:

U.S.	800-INTEL4U
U.K.	(0) 793 / 641469
France	(16) 1 / 30.57.70.00
Germany	(0) 89 / 903-2025
Netherlands	(0) 10 / 4.07.11.30
Italy	(0) 2 / 892.00950
Israel	(0) 3 / 548-3222
Sweden	(0) 8 / 825416
Finland	(9) 0 / 544644
Spain	(9) 1 / 308-2552
Switzerland	call Germany

Denmark	call Sweden
Norway	call Sweden
Other countries	call nearest facility

Please have your registration card available at the time of the call. In addition, it is helpful to have the system hardware configuration written down including internal devices and add-in cards. Manufacturer, model number and revision level are all very useful.

Included in the Motif product is a product registration card. To register your software, this card should be filled out and returned to Intel by mail.

By registering your software products, you will receive access to Intel's bulletin board and information regarding Intel's UNIX training. Dial the bulletin board in the U.S. at (503) 640-3042. The bulletin board supports up to 2400 baud modems and contains the latest product information, technical articles and answers to frequently asked questions.

The initial session on the bulletin board is restricted to public access. Prior to logging out of the bulletin board for the first time, please leave a message for the System Operator to upgrade your privilege level to Installation Level Support. Please include your product registration number with your request. Your access privileges will be upgraded by the next business day. After 30 days your ILS access privileges will expire.

Contact your Intel field salesperson or Intel distributor for information on other Intel UNIX software products.

* UNIX is a registered trademark and OPEN LOOK is a trademark of AT&T. X Window System is a trademark of Massachusetts Institute of Technology. Motif is a trademark of the Open Software Foundation

intel

product release notes

Intel Motif[1]

Intel UNIX[2] System V

Release 4.0

1 Motif is a registered trademark of Open Software Foundation, Inc.

2 UNIX is a registered trademark of AT&T.

DOCUMENT 466800

SHEET 1 REV A

IDENTIFICATION

Name: Intel Motif Version 1.0.A

PRODUCT CONTENTS

Intel Motif version 1.0.A, contains *mwm, uil,* and the Motif development libraries and include files for Intel X11 release 3. The product is provided on both 1.44 MB 3.5 inch and 1.2 MB 5.25 inch floppies. These release notes contain installation instructions, and some general usage information. Motif programming documentation is also available from Intel by ordering product code MOTIFPROGDOC.

SUPPORTED HARDWARE/SOFTWARE

Intel Motif 1.0.A runs on UNIX System V Release 4.0 running the Intel X11 release 3. Prerequisites for Intel Motif version 1.0.A include the following Intel X11 packages:

- Graphics End User System
- Graphics Programmers Package

Motif will operate properly on all of the hardware supported by Intel's X11 product.

INSTALLATION

If you have a prior version of Motif installed on your system, you must remove it (use *pkgrm*) before installing this version of Motif. Intel Motif is installed using the *pkgadd* utility. Login as root, type *pkgadd -d diskette/1|2/ motif,* and follow the prompts. Installation does not require a kernel rebuild. Type *pkginfo* after installation to confirm a successful installation.

REMOVAL

pkgrm will remove Intel Motif, restore any files that have been overlaid, and remove any directories it empties.

COMPILATION ENVIRONMENTS

Intel Motif installation will replace and save a few Intel X11 libraries and include files. Specifically, the Toolkit intrinsics (/usr/X/lib/libXt.a), the Athena widgets (/usr/X/lib/libXaw.a), HP widgets (/usr/X/lib/libXw.a), and related intrinsic include files will be replaced.

Before installing Motif, your system is configured for the "default compilation environment", which is appropriate for compiling OPEN LOOK[3] applications.

3. OPEN LOOK is a trademark of AT&T.

After installing MOTIF, your system is configured for the "Motif compilation environment", which is necessary for compiling Motif applications. Using the "Motif compilation environment" to compile an OpenLook application or vice-versa will produce unpredictable results.

Two scripts are provided that allow you to identify and change environments between Motif and OPEN LOOK. The first script, **whichXt**, allows you to identify the current compilation environment. **whichXt** takes no parameters and prints one of the following messages:

- "Default environment installed." (OpenLook compilation environment), or
- "Motif environment installed." (Intel Motif compilation environment).

The command **Xtchangeto** allows you to change your environment. This command must be executed as root. The command: **Xtchangeto Default** changes the compilation environment from Motif to the default (OPENLOOK). The command: **Xtchangeto Motif** changes the compilation environment from the default (OPENLOOK) to Motif. If you type **Xtchangeto Motif** while the "Motif environment" is installed, then no action takes place.

The *pkgrm* command will work correctly regardless of the current compilation environment.

Demo Programs

Source code for some example Motif applications is provided with this release. These programs are not supported by Intel, but are provided as an example of how to write Motif applications. These programs are also referred to by the Motif documentation. They are installed in the directory **/usr/src/motif/examples**.

UIL Compiler

This version of Intel Motif contains a filter that should be run on any UIL files which were written for a release of Motif prior to version 1.0.A. The filter converts Motif UIL language names to their final form. The filter replaces all instances of the old names in the UIL code. Occurrences in quoted strings and comments are not modified.
To run the filter (for example, on sourcefile.uil):

uilfilter < sourcefile.uil > sourcefile.newuil

mv sourcefile.newuil sourcefile.uil

CUSTOMIZATION

The operation of Motif is controlled via the server's resource database and the contents of a .mwmrc file in your home directory. By default, resources are fetched from /usr/X/lib/app-defaults/Mwm —you can override these resources with your own Xdefaults file. Menus, button bindings, and key bindings are defined by /usr/X/lib/system.mwmwrc — you can override these with your own .mwmrc file (see pages 5-46 of this release).

The resources that are loaded for Motif from the /usr/X/lib/app-defaults/Mwm.rdb file have definitions that will properly support monochrome

or high or low resolution color. The defaults can be modified on a system level by editing the Mwm.rdb file or on a user level by adding entries to the Xdefaults file.

HELPFUL HINTS

A resource "Mwm*useQuitQuery" has been added to Mwm. This is a boolean resource which tells mwm whether to display the "Mwm Quit?" dialog box before exiting. This resource is set to False in /usr/X/lib/app-defaults/Mwm.

COMPATIBILITY

This release of Motif (MOTIF40R1.0) is the same product as the Motif released for Unix V.3.2 (MOTIF32R1.0). The only difference is that Motif40R1.0 has been built as a Unix V.4 package. Therefore when migrating Motif programs from 3.2 to 4.0, no adjustments to the Motif portion of the program should be necessary.

REFERENCES

The "*OSF/Motif User's Guide*" will get you started but for more detailed information on *mwm* and programming with the Motif widget set, Intel sells a product that includes a *Motif Style Guide, Motif Programmer's Guide, Motif Programmer's Reference,* and *The Motif Application Environment Specification.* The product code is MOTIF32PDOC and is available from the same organization that provided the Motif software product.

MWM RESOURCE REFERENCE

This section describes the various resources you can set in your `.Xdefaults` file to configure the appearance and behavior of mwm.

The .Xdefaults File

The file `.Xdefaults` in your home directory specifies the resources that control mwm's appearance and behavior. mwm searches the following locations for resources:

1. The server's resource database, if any (see the discussion of "Using `xrdb`" later in this release note).
2. Your `.Xdefaults` file, if any (or the file specified by the `$XENVIRONMENT` shell variable, if different).
3. The default resource file `/usr/X/lib/app-defaults/Mwm`.

The basic syntax of each line of the `.Xdefaults` file is:

*client[*part...]*resource*: *value*

For mwm, the *client* is always mwm or Mwm. (This document uses the client *class* Mwm (capitalized) rather than the client *name* mwm (lowercase).) The *parts* that you can use depend on the resource (as described later in this section), and the *resource* name is one of the names listed in the following pages. The *value* is separated from the resource name by a colon and any amount of white space; it must be of an appropriate type for the resource. For example, the line:

```
Mwm*menu*background:  CadetBlue
```

sets the background color for all mwm menus to cadet blue.

In general, the order of lines in the `.Xdefaults` file does not matter. Exception: if there are two lines that define different values for the same resource, the second line overrides the first.

Blank lines and lines beginning with an exclamation point (`!`) in the `.Xdefaults` file are ignored.

You must restart mwm for changes in the `.Xdefaults` file to take effect.

Using xrdb

If you have used the `xrdb` command to load your `.Xdefaults` file into the window server's resource database, you must use the command

```
$ xrdb -merge $HOME/.Xdefaults
```

after you change the `.Xdefaults` file and before you restart `mwm`. This command updates the resource database with your new values.

If you *remove* a resource from your `.Xdefaults` file, you must clear the resource database (using `xrdb -remove`) and then re-load any system resource files before you merge in your `.Xdefaults` file with `xrdb -merge`. If you don't do this, the removed resource will retain its previous value. (See the `xrdb` manpage for more information.)

Resource Classes

Every resource belongs to a *resource class*. If a resource does not have an explicit value, it "inherits" the value of the class to which it belongs. This lets you set the values of several related resources with a single line in your `.Xdefaults` file.

Resource names always begin with a lowercase letter, and resource class names always begin with an uppercase letter. Most `mwm` resources are the only members of their class (for example, the resource `windowMenu` is the only resource of class `WindowMenu`), but a few are members of larger classes (notably `Foreground` and `Background`).

For example, the resource class `Background` includes the resources `background`, `backgroundPixmap`, `activeBackground`, `activeTopShadowColor`, `topShadowColor`, `iconImageBackground`, `iconImageTopShadowColor`, `matteBackground`, and `matteTopShadowColor`, so you can set all these resources to the color aquamarine with the `.Xdefaults` line

```
Mwm*Background: aquamarine
```

A specific resource specification overrides the resource class specification for that resource. For example, if you added the line

```
Mwm*iconImageBackground: green
```

to your `.Xdefaults` file, then icon image backgrounds would be green and all other backgrounds would remain aquamarine.

Resource Types

mwm has three types of resources: *general*, *component-specific*, and *client-specific*. Each affects a different part of mwm and has a slightly different syntax from that of the other two.

General Resources

General mwm resources specify the way the window manager looks and acts. The syntax of general resources is:

```
Mwm*resource: value
```

For example, the line

```
Mwm*keyboardFocusPolicy: pointer
```

makes the keyboard focus go to the window that currently contains the pointer.

Component-Specific Resources

Component-specific resources specify the appearance of window frames, icons, menus, and the feedback window. The syntax of component appearance resources is:

```
Mwm*[component*]resource: value
```

where the optional *component* part specifies the component whose appearance is affected by the resource. The available components are:

`client`	Client window frames
`client*title`	Title area of client window frames (including the menu button, minimize button, and maximize button)
`icon`	Icons (minimized windows)
`menu`	Menus
`menu*`*menu_name*	The specified menu (the *menu_name* is the name of the menu in the `.mwmrc` file)
`feedback`	The feedback window (the small indicator that appears when a window is being moved or resized)

If the *component** part is omitted, the resource specifies the appearance of *all* mwm components.

For example, the line

```
Mwm*client*background: green
```

makes the background color of all window frames green; the line

```
Mwm*background: green
```

makes the background color of *all* mwm components green.

Resources that affect a particular component override resources that affect all components. For example, suppose your `.Xdefaults` file contains the following lines:

```
Mwm*background:                 red
Mwm*menu*background:            blue
Mwm*menu*RootMenu*background: yellow
```

In this case, the root menu (but not its submenus) will have a yellow background. All menus but the root menu will have a blue background, and all other mwm components will have red backgrounds. The order of the lines in the `.Xdefaults` file does not matter.

Client-Specific Resources

Client-specific resources specify the way the window manager treats windows and icons belonging to specific clients. The syntax of client-specific resources is as follows:

`Mwm*`*`[client*]resource`*`:` *value*

where the optional *client* part is a client name (for example, `xterm`), a client class (for example, `XTerm`), or the word `default` to apply to clients with *no* name or class (as determined by the window's `WM_CLASS` property). If the *client** part is omitted, the resource applies to all clients.

For example, the line

```
Mwm*xterm*focusAutoRaise: True
```

makes any xterm window automatically raise itself to the top of the stack when it gets the keyboard focus; the line

```
Mwm*focusAutoRaise: True
```

makes *all* windows raise themselves when they get the keyboard focus.

Note that some windows have names that cannot be used in resource specifications. For example, you cannot set a client-specific resource for a window whose name contains a space or other special character.

Resources that affect a particular client override resources that affect all clients. For example, suppose your `.Xdefaults` file contains the following lines:

```
Mwm*focusAutoRaise:          True
Mwm*xclock*focusAutoRaise: False
```

In this case, all windows *except* `xclock` windows will autoraise.

MWM Resource Descriptions

The following pages describe all of `mwm`'s resources in alphabetical order.

Note: Despite the name, the value of a "pixmap" resource must be a *bitmap* file, not a pixmap file. (`mwm` does not understand the pixmap format used by files in `/usr/X/include/X11/pixmaps`.) To specify a bitmap file, you can use either an absolute pathname or a simple filename (in which case `mwm` looks for the file in the directory specified by the `bitmapDirectory` resource).

activeBackground

Class: `Background`

Syntax: `Mwm*`*`[component*]`*`activeBackground:` *color*

Default: Based on screen type

Meaning: Background color for the window frame or icon with the keyboard focus. (Does not apply to menus or the feedback window.)

activeBackgroundPixmap

Class: `BackgroundPixmap`

Syntax: `Mwm*`*`[component*]`*`activeBackgroundPixmap:` *bitmap_file*

Default: Based on screen type

Meaning: Background bitmap for the window frame or icon with the keyboard focus. (Does not apply to menus or the feedback window.)

activeBottomShadowColor

Class: `Foreground`

Syntax: `Mwm*[component*]activeBottomShadowColor:` *color*

Default: Based on screen type

Meaning: Color for the bottom and right-hand bevels of the window frame or icon with the keyboard focus. (Does not apply to menus or the feedback window.)

activeBottomShadowPixmap

Class: `BottomShadowPixmap`

Syntax: `Mwm*[component*]activeBottomShadowPixmap:` *bitmap_file*

Default: Based on screen type

Meaning: Bitmap for the bottom and right-hand bevels of the window frame or icon with the keyboard focus. (Does not apply to menus or the feedback window.)

activeForeground

Class: `Foreground`

Syntax: `Mwm*[component*]activeForeground:` *color*

Default: Based on screen type

Meaning: Foreground color for the window frame or icon with the keyboard focus. (Does not apply to menus or the feedback window.)

activeTopShadowColor

Class: `Background`

Syntax: `Mwm*[component*]activeTopShadowColor:` *color*

Default: Based on screen type

Meaning: Color for the top and left-hand bevels of the window frame or icon with the keyboard focus. (Does not apply to menus or the feedback window.)

activeTopShadowPixmap

Class: TopShadowPixmap

Syntax: Mwm*[component*]activeTopShadowPixmap: *bitmap_file*

Default: Based on screen type

Meaning: Bitmap for the top and left-hand bevels of the window frame or icon with the keyboard focus. (Does not apply to menus or the feedback window.)

autoKeyFocus

Class: AutoKeyFocus

Syntax: Mwm*autoKeyFocus: *boolean*

Default: True

Meaning: If true, when the window with the keyboard focus is iconified, the focus goes to the previous window that had it. If false, the focus remains with the iconified window. (Only meaningful if keyboardFocusPolicy is explicit.)

autoRaiseDelay

Class: AutoRaiseDelay

Syntax: Mwm*autoRaiseDelay: *time_in_milliseconds*

Default: 500

Meaning: Time the pointer can remain in a window before the window is automatically raised. (Only meaningful if focusAutoRaise is true and keyboardFocusPolicy is pointer.)

background

Class: Background

Syntax: Mwm*[component*]background: *color*

Default: Based on screen type

Meaning: Background color of the specified component.

backgroundPixmap

Class: BackgroundPixmap

Syntax: Mwm*[component*]backgroundPixmap: *bitmap_file*

Default: Based on screen type

Meaning: Background bitmap of the specified component.

bitmapDirectory

Class: BitmapDirectory

Syntax: Mwm*bitmapDirectory: *directory_pathname*

Default: /usr/include/X11/bitmaps

Meaning: Directory searched for any bitmap file that is specified by filename only (without an explicit pathname).

bottomShadowColor

Class: Foreground

Syntax: Mwm**[component*]*bottomShadowColor: *color*

Default: Based on screen type

Meaning: Color for the bottom and right-hand bevels of the specified component.

bottomShadowPixmap

Class: BottomShadowPixmap

Syntax: Mwm**[component*]*bottomShadowPixmap: *bitmap_file*

Default: Based on screen type

Meaning: Bitmap for the bottom and right-hand bevels of the specified component.

buttonBindings

Class: ButtonBindings

Syntax: Mwm*buttonBindings: *string*

Default: None

Meaning: Name of a set of button bindings in the .mwmrc file. The named set of button bindings is merged with the default button bindings.

cleanText

Class: CleanText

Syntax: Mwm*cleanText: *boolean*

Default: True

Meaning: If true, text in window titles and feedback windows is surrounded by a rectangle of the background color. If false, text is drawn directly on the background bitmap. (Only meaningful if there is a background bitmap.)

clientAutoPlace

Class: ClientAutoPlace

Syntax: Mwm*clientAutoPlace: *boolean*

Default: True

Meaning: If true, a newly-created window is offset below and to the right of the last window created. If false, the window is created at its default position. In either case, a user-specified geometry specification takes precedence.

clientDecoration

Class: ClientDecoration

Syntax: Mwm*[client*]clientDecoration: *string*

Default: all

Meaning: The set of Motif decorations applied to the specified client's windows. The value of this resource is a space-separated series of decoration names, each of which can be preceded by a sign (+ or –). The available decorations are:

border	Outer window border
title	Title area
menu	Menu button (implies title)
minimize	Minimize button (implies title)
maximize	Maximize button (implies title)
resizeh	Resize handles (implies border)
none	No decorations
all	All decorations

A decoration name preceded by a plus sign or no sign is added to the list of decorations. A decoration name preceded by a minus sign is removed from the list of decorations. If the first decoration name in the list is preceded by a plus sign or no sign, the list of decorations starts empty (subsequent entries are added to it). If the first decoration name is preceded by a minus sign, the list of decorations starts off with all decorations (subsequent entries are removed from it). For example:

```
Mwm*xclock*clientDecoration: +resizeh
Mwm*xlogo*clientDecoration: -minimize
```

This gives xclock windows borders and resize handles only; xlogo windows get all decorations but the minimize button.

Even when a window does not have a decoration, the corresponding function is still available through the window menu. (If the menu button is not there, you can get the window menu by pressing **<Alt-space>** or **<Shift-Esc>**.) To make the function unavailable, use the `clientFunctions` resource.

clientFunctions

Class: `ClientFunctions`

Syntax: `Mwm*[client*]clientFunctions:` *string*

Default: `all`

Meaning: The set of window manager functions available for the specified client's windows. Functions that are not available for a window cannot be performed on the window using the window frame, window menu, or keyboard shortcuts; the corresponding decorations do not appear on the window frame, and the corresponding menu items do not appear on the window menu.

The value of this resource is a space-separated series of function names, each of which *must* be preceded by a sign (+ or –). The available functions are:

resize	Resize the window (f.resize)
`move`	Move the window (`f.move`)
`minimize`	Minimize the window (`f.minimize`)
`maximize`	Maximize the window (`f.maximize`)
`close`	Close the window (`f.kill`)
`none`	No functions
`all`	All functions

A function name preceded by a plus sign is added to the list of functions; a function name preceded by a minus sign is removed from the list of functions. If the first function name in the list is preceded by a plus sign, the list of functions starts empty (subsequent entries are added to it). If the first function name is preceded by a minus sign, the list of functions starts off with all functions (subsequent entries are removed from it). For example:

```
Mwm*xclock*clientDecoration: +move
Mwm*xlogo*clientDecoration: -maximize
```

This makes `xclock` windows movable, but not anything else; `xlogo` windows can be resized, moved, minimized, or closed, but not maximized.

Note that if the resources

```
Mwm*xlogo*clientDecoration: -minimize
```

and

```
Mwm*xlogo*clientDecoration: -maximize
```

are specified at the same time, `xlogo` windows will not have either a minimize or a maximize button. They can be minimized using the window menu, but they cannot be maximized by any means.

colormapFocusPolicy

Class: `ColormapFocusPolicy`

Syntax: `Mwm*colormapFocusPolicy:` *string*

Default: `keyboard`

Meaning: Policy that determines which window has the *colormap focus*. The colormap of the window with the colormap focus is loaded into the server, and is used for all other windows. The value of this resource must be one of the following words:

`keyboard` The window with the keyboard focus also has the colormap focus.

`pointer` The window that currently contains the pointer has the colormap focus (even if that window does not have the keyboard focus).

`explicit` The last window that was the subject of the function `f.focus_color` has the colormap focus.

configFile

Class: `ConfigFile`

Syntax: `Mwm*configFile:` *pathname*

Default: `$HOME/.mwmrc` if it exists, otherwise `/usr/X/lib/system.mwmrc`

Meaning: Pathname of the `.mwmrc` file. This file provides definitions for mwm's menus, key bindings, and button bindings.

deiconifyKeyFocus

Class: DeiconifyKeyFocus

Syntax: Mwm*deiconifyKeyFocus: *boolean*

Default: True

Meaning: If true, a window automatically receives the keyboard focus when it is restored from a minimized state. If false, the window with the keyboard focus retains it. (Only meaningful if keyboardFocusPolicy is explicit.)

doubleClickTime

Class: DoubleClickTime

Syntax: Mwm*doubleClickTime: *time_in_milliseconds*

Default: 500

Meaning: The length of time that can elapse between the two clicks of the mouse in a double-click, in milliseconds.

enforceKeyFocus

Class: EnforceKeyFocus

Syntax: Mwm*enforceKeyFocus: *boolean*

Default: True

Meaning: If true, the keyboard focus is explicitly set to the window that should have it. If false, the keyboard focus is *not* explicitly set to windows that are "globally active" input windows (such as a scroll bar that can be operated from the keyboard without setting the focus to the associated client window).

fadeNormalIcon

Class: FadeNormalIcon

Syntax: Mwm*fadeNormalIcon: *boolean*

Default: False

Meaning: If true, the icon representing a normal (not iconified) window in the icon box is "grayed out" with a stipple pattern. If false, icons representing normal windows are not grayed out. (Only meaningful if useIconBox is true.)

focusAutoRaise

Class: FocusAutoRaise

Syntax: Mwm*[*client**]focusAutoRaise: *boolean*

Default: True

Meaning: If true, the specified client's windows are automatically raised to the top of the window stack when it gets the keyboard focus (after a delay specified by the autoRaiseDelay resource). If false, windows retain their place in the window stack whether or not they have the keyboard focus.

fontList

Class: FontList

Syntax: Mwm*[*component**]fontList: *font_specification*

Default: fixed

Meaning: Font used for text drawn by mwm in the specified component. Note that you can set the fonts used in the various components separately:

Mwm*fontList — All text

Mwm*client*fontList — Titles of client windows

Mwm*icon*fontList — Labels under icons

Mwm*menu*fontList — All menus

Mwm*menu**menu_name**fontList — The specified menu

Mwm*feedback*fontList — The feedback window

The font specification can include the "wild card" characters * and ? (for example, *-helvetica-*-24-*). If you use wild card characters, the font used is the first font that matches the specified pattern.

foreground

Class: Foreground

Syntax: Mwm*[*component**]foreground: *color*

Default: Based on screen type

Meaning: Foreground color of the specified component.

frameBorderWidth

Class: FrameBorderWidth

Syntax: Mwm*frameBorderWidth: *width_in_pixels*

Default: 5

Meaning: The width of a window border *without* resize handles (including the "3-D" shadows), in pixels . (See resizeBorderWidth).

iconAutoPlace

Class: IconAutoPlace

Syntax: Mwm*iconAutoPlace: *boolean*

Default: True

Meaning: If true, when a window is minimized its icon is placed as specified by the iconPlacement resource. If false, when a window is minimized the icon is placed at the upper left corner of the window. (Only meaningful if useIconBox is false.)

iconBoxGeometry

Class: IconBoxGeometry

Syntax: Mwm*iconBoxGeometry: *widthxheight[±x_offset±y_offset]*

Default: 6x1+0-0

Meaning: The size, shape, and position of the icon box, specified as a standard X Window System geometry specification. The *width* and *height* are given in icons and the *x_offset* and *y_offset* are in pixels, so the default value of 6x1+0-0 specifies an icon box that is big enough for six icons (six icons wide and one icon high) and is located in the lower left corner of the screen. (Only meaningful if useIconBox is true.)

iconBoxName

Class: IconBoxName

Syntax: Mwm*iconBoxName: *string*

Default: iconbox

Meaning: The name used when looking up resources for the icon box. Use this name with client-specific resources to specify the appearance and behavior of the icon box. (Only meaningful if useIconBox is true.)

iconBoxTitle

Class: IconBoxTitle

Syntax: Mwm*iconBoxTitle: *string*

Default: Icons

Meaning: The title that appears in the title bar of the icon box. (Only meaningful if useIconBox is true.)

iconClick

Class: IconClick

Syntax: Mwm*iconClick: *boolean*

Default: True

Meaning: If true, clicking once on an icon causes the window menu to pop up and stay up. If false, clicking once on an icon has no effect (to restore the icon to a window, double-click on it; to get the window menu, move the pointer into the icon and press <Alt-space>). (Only meaningful if useIconBox is false.)

iconDecoration

Class: IconDecoration

Syntax: Mwm*iconDecoration: *string*

Default: Based on screen size

Meaning: The set of Motif decorations applied to icons. The value of this resource is a space-separated series of decoration names. The available decorations are:

image — Icon image (picture)

label — Icon label (text under picture)

activelabel — Label that is not truncated to the width of the icon when the icon is selected

For example, the line

```
Mwm*iconDecoration: label
```

makes icons consist of a label part only (this is the default for small screens); the line

```
Mwm*iconDecoration: label image activelabel
```

makes icons consist of a picture with a label underneath; the label expands to its full width when the icon is selected. (This is the default for large screens.)

iconImage

Class: IconImage

Syntax: Mwm*[client*]iconImage: *bitmap_file*

Default: Picture of four small boxes

Meaning: Bitmap used for icon image when the specified client's windows are iconified. (Only meaningful if iconDecoration includes image.)

iconImageBackground

Class: Background

Syntax: Mwm*[client*]iconImageBackground: *color*

Default: Same as icon background

Meaning: Color used for drawing background ("white") pixels of the specified client's icon image. (Only meaningful if iconDecoration includes image.)

iconImageBottomShadowColor

Class: Foreground

Syntax: Mwm*[client*]iconImageBottomShadowColor: *color*

Default: Same as icon bottom shadow

Meaning: Color of the bottom and right-hand bevels of the specified client's icon image. (Only meaningful if iconDecoration includes image.)

iconImageBottomShadowPixmap

Class: BottomShadowPixmap

Syntax: Mwm*[client*]iconImageBottomShadowPixmap: *color*

Default: Same as icon bottom shadow pixmap

Meaning: Bitmap for the bottom and right-hand bevels of the specified client's icon image. (Only meaningful if iconDecoration includes image.)

iconImageForeground

Class: Foreground

Syntax: Mwm*[client*]iconImageForeground: *color*

Default: Same as icon foreground

Meaning: Color used for drawing foreground ("black") pixels of the specified client's icon image. (Only meaningful if iconDecoration includes image.)

iconImageMaximum

Class: IconImageMaximum

Syntax: Mwm*iconImageMaximum: *widthxheight*

Default: 50x50

Meaning: Maximum size (in pixels) of an icon image. Images that are smaller than the specified size are centered within the icon area; images large than the specified size are truncated on the bottom and right side. Must be smaller than 128x128. (Only meaningful if iconDecoration includes image.)

iconImageMinimum

Class: IconImageMinimum

Syntax: Mwm*iconImageMinimum: *widthxheight*

Default: 32x32

Meaning: Minimum size (in pixels) of an icon image. Images that are smaller than the specified size are ignored (the default image is used instead). Must be larger than 16x16. (Only meaningful if iconDecoration includes image.)

iconImageTopShadowColor

Class: Background

Syntax: Mwm*[client*]iconImageTopShadowColor: *color*

Default: Same as icon top shadow color

Meaning: Color of the top and left-hand bevels of the specified client's icon image. (Only meaningful if iconDecoration includes image.)

iconImageTopShadowPixmap

Class: TopShadowPixmap

Syntax: Mwm*[client*]iconImageTopShadowPixmap: *color*

Default: Same as icon top shadow pixmap

Meaning: Bitmap for the top and left-hand bevels of the specified client's icon image. (Only meaningful if iconDecoration includes image.)

iconPlacement

Class: IconPlacement

Syntax: Mwm*iconPlacement: *string*

Default: left bottom

Meaning: Policy for placing icons on the screen. The value of this resource consists of two words chosen from the following list, separated by a space:

left	Left-to-right
right	Right-to-left
top	Top-to-bottom
bottom	Bottom-to-top

If the first word is horizontal (left or right), the second must be vertical (top or bottom); if the first word is vertical (top or bottom), the second must be horizontal (left or right).

The first word determines the direction the icons are laid out; the second word determines the edge of the screen against which they are placed. For example:

```
Mwm*iconPlacement: left bottom
```

This line causes icons to be placed from left to right at the bottom of the screen. The icon of the first window to be minimized is placed in the lower left corner of the screen; the next icon is placed to its right, and so on.

The iconPlacement resource is meaningful only if iconAutoPlace is true and useIconBox is false. (If useIconBox is true, the icon box always uses the placement policy left top.)

iconPlacementMargin

Class: IconPlacementMargin

Syntax: Mwm*iconPlacementMargin: *distance_in_pixels*

Default: Varies

Meaning: Minimum distance between the edge of the screen and the edge of an icon, in pixels.

interactivePlacement

Class: InteractivePlacement

Syntax: Mwm*interactivePlacement: *boolean*

Default: False

Meaning: If true, when a window is created without a geometry specification you are prompted to place it. (An outline of the new window appears on the screen: move the outline to the desired position and press and hold the left mouse button, then drag the mouse to select the new window's size and shape.) If false, when a window is created without a geometry specification it is placed as specified by the clientAutoPlace resource.

keyBindings

Class: KeyBindings

Syntax: Mwm*keyBindings: *string*

Default: Standard key bindings

Meaning: Name of a set of key bindings in the .mwmrc file. The named set of key bindings replaces the default key bindings.

keyboardFocusPolicy

Class: KeyboardFocusPolicy

Syntax: Mwm*keyboardFocusPolicy: *string*

Default: explicit

Meaning: Policy that determines which window has the keyboard focus. The value of this resource must be one of the following words:

- `pointer` You give a window the keyboard focus by moving the pointer into the window (or onto its frame).
- `explicit` You give a window the keyboard focus by clicking anywhere in the window or on its frame with the left mouse button, or by using **<Alt-Tab>** to move the keyboard focus from one window to another.

limitResize

Class: LimitResize

Syntax: Mwm*limitResize: *boolean*

Default: True

Meaning: If true, you are not allowed to resize a window larger than the size set by the maximumMaximumSize resource. If false, you can make a window as large as you want.

lowerOnIconify

Class: LowerOnIconify

Syntax: Mwm*lowerOnIconify: *boolean*

Default: True

Meaning: If true, when a window is iconified, the icon is sent behind all other windows. If false, when a window is iconified, the icon has the same place in the window stacking order as the window did.

matteBackground

Class: Background

Syntax: Mwm**[client*]*matteBackground: *color*

Default: Same as background

Meaning: Background color of the specified client's matte. (Only meaningful if matteWidth is greater than 0.)

matteBottomShadowColor

Class: Foreground

Syntax: Mwm*[client*]matteBottomShadowColor: *color*

Default: Same as bottom shadow color

Meaning: Color for the bottom and right-hand bevels of the specified client's matte. (Only meaningful if matteWidth is greater than 0.)

matteBottomShadowPixmap

Class: BottomShadowPixmap

Syntax: Mwm*[client*]matteBottomShadowPixmap: *color*

Default: Same as bottom shadow pixmap

Meaning: Bitmap for the bottom and right-hand bevels of the specified client's matte. (Only meaningful if matteWidth is greater than 0.)

matteForeground

Class: Foreground

Syntax: Mwm*[client*]matteForeground: *color*

Default: Same as foreground

Meaning: Foreground color of the specified client's matte. (Only meaningful if matteWidth is greater than 0.)

matteTopShadowColor

Class: Background

Syntax: Mwm*[client*]matteTopShadowColor: *color*

Default: Same as top shadow color

Meaning: Color for the top and left-hand bevels of the specified client's matte. (Only meaningful if matteWidth is greater than 0.)

matteTopShadowPixmap

Class: TopShadowPixmap

Syntax: Mwm*[client*]matteTopShadowPixmap: *color*

Default: Same as top shadow pixmap

Meaning: Bitmap for the top and left-hand bevels of the specified client's matte. (Only meaningful if matteWidth is greater than 0.)

matteWidth

Class: `MatteWidth`

Syntax: `Mwm*`*`[client*]`*`matteWidth:` *width_in_pixels*

Default: 0

Meaning: Width of the specified client's matte. (The *matte* is an optional blank area between the edge of the working part of a window and the frame. It does not have any function.)

maximumClientSize

Class: `MaximumClientSize`

Syntax: `Mwm*`*`[client*]`*`maximumClientSize:` *widthxheight*

Default: Full screen

Meaning: Size of the specified client's windows when maximized. The units of *width* and *height* are the units used by the client (characters `x` rows for `xterm`, pixels for most other clients).

maximumMaximumSize

Class: `MaximumMaximumSize`

Syntax: `Mwm*maximumMaximumSize:` *widthxheight*

Default: Twice the screen width and height

Meaning: Maximum permissible size for all windows. (Only meaningful if `limitResize` is true.)

moveThreshold

Class: `MoveThreshold`

Syntax: `Mwm*moveThreshold:` *distance_in_pixels*

Default: 4

Meaning: Distance (in pixels) you must move the mouse when dragging or resizing a window or icon before the window or icon begins to move. This threshold prevents movement if the mouse moves a tiny bit during a click or double-click.

passButtons

Class: PassButtons

Syntax: Mwm*passButtons: *boolean*

Default: False

Meaning: If true, when you press a button that invokes a window manager function while the pointer is in a client window, the window manager function is performed *and* the button-press event is passed to the client. If false, the window manager function is performed and the button-press event is "swallowed" by mwm. (Only meaningful if the button bindings specified by the buttonBindings resource specify buttons that have an effect in the window context.)

passSelectButton

Class: PassSelectButton

Syntax: Mwm*passSelectButton: *boolean*

Default: True

Meaning: If true, when you press the left mouse button in a window to give the window the keyboard focus, the button-press event is passed to the client. If false, the button-press event is "swallowed" by mwm. (Only meaningful if keyboardFocusPolicy is explicit.)

positionIsFrame

Class: PositionIsFrame

Syntax: Mwm*positionIsFrame: *boolean*

Default: True

Meaning: If true, window geometry specifications specify the location of the upper left corner of the window frame. If false, window geometry specifications specify the location of the upper left corner of the working area of the window (the *inner* edge of the window frame).

positionOnScreen

Class: PositionOnScreen

Syntax: Mwm*positionOnScreen: *boolean*

Default: True

Meaning: If true, a newly-created window will always be positioned so that no part of the window is off the screen. (Windows larger than the screen are positioned so the upper left corner is on the screen.) If false, a newly-created window will be positioned wherever its geometry specification requests, even if that is completely off the screen.

quitTimeout

Class: QuitTimeout

Syntax: Mwm*quitTimeout: *time_in_milliseconds*

Default: 1000

Meaning: Maximum time (in milliseconds) that mwm will wait for a client to respond to the WM_SAVE_YOURSELF message issued by the f.kill function. (Only significant if the client has declared that it expects messages of this type.)

resizeBorderWidth

Class: ResizeBorderWidth

Syntax: Mwm*resizeBorderWidth: *width_in_pixels*

Default: 10

Meaning: The width of a window border *with* resize handles (including the "3-D" shadows), in pixels . (See frameBorderWidth).

resizeCursors

Class: ResizeCursors

Syntax: Mwm*resizeCursors: *boolean*

Default: True

Meaning: If true, the cursor changes to an "arrow-in-a-corner" or an "arrow-with-line" when the pointer is in one of the resize areas of a window. If false, the cursor is a simple arrow when the pointer is in any part of the frame.

saveUnder

Class: SaveUnder

Syntax: Mwm*[*component**]saveUnder: *boolean*

Default: False

Meaning: If true, mwm requests "save unders" for the window frame of any client window that has requested "save unders" for itself. (The "save under" property requests the window server to save the contents of a window when it is obscured, which speeds redisplay when the obscured portion is uncovered.) If false, mwm does not request "save unders" for any window frame. (Only meaningful if the server implements save unders.)

showFeedback

Class: ShowFeedback

Syntax: Mwm*showFeedback: *string*

Default: all

Meaning: The set of mwm operations that show or require feedback. The value of this resource is a space-separated series of operation names. Feedback is given for operations named in this list; no feedback is given for operations not named in this list. The available operations are:

move	Show window position while a window is being moved.
resize	Show window size while a window is being resized.
placement	Show window size and position while a newly-created window is being placed (only meaningful if interactivePlacement is true).
behavior	Confirm behavior switch (<Ctrl-Shift-Alt-!>) with "toggle behavior?" dialog.
restart	Confirm window manager restart with "restart Mwm?" dialog.
none	Don't show feedback on any of these operations.
all	Show feedback on all operations.

For example:

```
Mwm*showFeedback: move resize behavior
```

This makes mwm show a window's position when it is being moved and its size when it is being resized, and makes it confirm a behavior switch, but no confirmation is required to restart mwm and a newly-created window's size and position are not shown if `interactivePlacement` is true.

startupKeyFocus

Class: `StartupKeyFocus`

Syntax: `Mwm*startupKeyFocus:` *boolean*

Default: `True`

Meaning: If true, a newly-created window automatically gets the keyboard focus. If false, the keyboard focus stays where it was when a window is created. (Only meaningful if `keyboardFocusPolicy` is `explicit`.)

topShadowColor

Class: `Background`

Syntax: `Mwm*`*[component*]*`topShadowColor:` *color*

Default: Based on screen type

Meaning: Color for the top and left-hand bevels of the specified component.

topShadowPixmap

Class: `TopShadowPixmap`

Syntax: `Mwm*`*[component*]*`topShadowPixmap:` *bitmap_file*

Default: Based on screen type

Meaning: Bitmap for the top and left-hand bevels of the specified component.

transientDecoration

Class: `TransientDecoration`

Syntax: `Mwm*transientDecoration:` *string*

Default: `menu title resizeh`

Meaning: The set of Motif decorations applied to "transient" (temporary) windows, such as popup windows. The syntax of this resource is the same as the `clientDecoration` resource.

transientFunctions

Class: TransientFunctions

Syntax: Mwm*transientFunctions: *string*

Default: -minimize -maximize

Meaning: The set of Motif functions available for transient windows. The syntax of this resource is the same as the clientFunctions resource.

useClientIcon

Class: UseClientIcon

Syntax: Mwm*[client*]useClientIcon: *boolean*

Default: False

Meaning: If true, an icon image provided by the specified client takes precedence over an icon image specified by the iconImage resource. If false, the iconImage resource takes precedence over an icon image provided by the client.

useIconBox

Class: UseIconBox

Syntax: Mwm*useIconBox: *boolean*

Default: False

Meaning: If true, all mwm icons are placed in a special *icon box window*. This window displays an icon for every window on the screen; icons for windows that are not currently iconified are "grayed-out" (if the fadeNormalIcon resource is true). If false, icons are placed on the root window along with all other windows.

useQuitQuery

Class: UseQuitQuery

Syntax: Mwm*useQuitQuery: *boolean*

Default: False

Meaning: If true, mwm confirms the function f.quit_mwm with the "Quit Mwm?" dialog. If false, mwm quits without asking for confirmation. (This resource is an Intel extension.)

wMenuButtonClick

Class: WMenuButtonClick

Syntax: Mwm*wMenuButtonClick: *boolean*

Default: True

Meaning: If true, clicking on the menu button in a window frame causes the window menu to pop up and stay up. If false, the menu stays up only as long as the mouse button is held down.

wMenuButtonClick2

Class: WMenuButtonClick2

Syntax: Mwm*wMenuButtonClick2: *boolean*

Default: True

Meaning: If true, double-clicking on the menu button in a window frame causes the window to be closed (killed). If false, double-clicking on the menu button just pops up the menu twice.

windowMenu

Class: WindowMenu

Syntax: Mwm*[*client*]windowMenu: *string*

Default: Standard window menu

Meaning: Name of a menu description in the .mwmrc file. The named menu is used as the window menu for the specified client.

Summary of MWM Resources

This table summarizes the class, value, and default of all of mwm's resources.

Resource	Class	Value	Default
activeBackground	Background	color	Based on screen type
activeBackgroundPixmap	BackgroundPixmap	bitmap file	Based on screen type
activeBottomShadowColor	Foreground	color	Based on screen type
activeBottomShadowPixmap	BottomShadowPixmap	bitmap file	Based on screen type
activeForeground	Foreground	color	Based on screen type
activeTopShadowColor	Background	color	Based on screen type
activeTopShadowPixmap	TopShadowPixmap	bitmap file	Based on screen type
autoKeyFocus	AutoKeyFocus	boolean	True
autoRaiseDelay	AutoRaiseDelay	time in milliseconds	500
background	Background	color	Based on screen type
backgroundPixmap	BackgroundPixmap	bitmap file	Based on screen type

Resource	Class	Value	Default
bitmapDirectory	BitmapDirectory	directory pathname	/usr/include/X11/bitmaps
bottomShadowColor	Foreground	color	Based on screen type
bottomShadowPixmap	BottomShadowPixmap	bitmap file	Based on screen type
buttonBindings	ButtonBindings	string	None
cleanText	CleanText	boolean	True
clientAutoPlace	ClientAutoPlace	boolean	True
clientDecoration	ClientDecoration	string	all
clientFunctions	ClientFunctions	string	all
colormapFocusPolicy	ColormapFocusPolicy	string	keyboard
configFile	ConfigFile	pathname	$HOME/.mwmrc or /usr/X/lib/system.mwmrc
deiconifyKeyFocus	DeiconifyKeyFocus	boolean	True
doubleClickTime	DoubleClickTime	time in milliseconds	500
enforceKeyFocus	EnforceKeyFocus	boolean	True
fadeNormalIcon	FadeNormalIcon	boolean	False
focusAutoRaise	FocusAutoRaise	boolean	True
fontList	FontList	font specification	fixed
foreground	Foreground	color	Based on screen type
frameBorderWidth	FrameBorderWidth	width in pixels	5
iconAutoPlace	IconAutoPlace	boolean	True
iconBoxGeometry	IconBoxGeometry	wxh[+x+y]	6x1+0-0
iconBoxName	IconBoxName	string	iconbox
iconBoxTitle	IconBoxTitle	string	Icons
iconClick	IconClick	boolean	True
iconDecoration	IconDecoration	string	Based on screen size
iconImage	IconImage	bitmap file	Picture of four small boxes
iconImageBackground	Background	color	Same as icon background
iconImageBottomShadowColor	Foreground	color	Same as icon bottom shadow
iconImageBottomShadowPixmap	BottomShadowPixmap	color	Same as icon bottom shadow pixmap
iconImageForeground	Foreground	color	Same as icon foreground
iconImageMaximum	IconImageMaximum	widthxheight	50x50
iconImageMinimum	IconImageMinimum	widthxheight	32x32
iconImageTopShadowColor	Background	color	Same as icon top shadow color
iconImageTopShadowPixmap	TopShadowPixmap	color	Same as icon top shadow pixmap
iconPlacement	IconPlacement	string	left bottom
iconPlacementMargin	IconPlacementMargin	distance in pixels	Varies
interactivePlacement	InteractivePlacement	boolean	False
keyBindings	KeyBindings	string	Standard key bindings
keyboardFocusPolicy	KeyboardFocusPolicy	string	explicit
limitResize	LimitResize	boolean	True
lowerOnIconify	LowerOnIconify	boolean	True

Resource	Class	Value	Default
matteBackground	Background	color	Same as background
matteBottomShadowColor	Foreground	color	Same as bottom shadow color
matteBottomShadowPixmap	BottomShadowPixmap	color	Same as bottom shadow pixmap
matteForeground	Foreground	color	Same as foreground
matteTopShadowColor	Background	color	Same as top shadow color
matteTopShadowPixmap	TopShadowPixmap	color	Same as top shadow pixmap
matteWidth	MatteWidth	width in pixels	0
maximumClientSize	MaximumClientSize	widthxheight	Full screen
maximumMaximumSize	MaximumMaximumSize	widthxheight	Twice the screen width and height
moveThreshold	MoveThreshold	distance in pixels	4
passButtons	PassButtons	boolean	False
passSelectButton	PassSelectButton	boolean	True
positionIsFrame	PositionIsFrame	boolean	True
positionOnScreen	PositionOnScreen	boolean	True
quitTimeout	QuitTimeout	time in milliseconds	1000
resizeBorderWidth	ResizeBorderWidth	width in pixels	10
resizeCursors	ResizeCursors	boolean	True
saveUnder	SaveUnder	boolean	False
showFeedback	ShowFeedback	string	all
startupKeyFocus	StartupKeyFocus	boolean	True
topShadowColor	Background	color	Based on screen type
topShadowPixmap	TopShadowPixmap	bitmap file	Based on screen type
transientDecoration	TransientDecoration	string	menu title resizeh
transientFunctions	TransientFunctions	string	-minimize -maximize
useClientIcon	UseClientIcon	boolean	False
useIconBox	UseIconBox	boolean	False
useQuitQuery	UseQuitQuery	boolean	False
wMenuButtonClick	WMenuButtonClick	boolean	True
wMenuButtonClick2	WMenuButtonClick2	boolean	True
windowMenu	WindowMenu	string	Standard window menu

MENUS, BUTTON BINDINGS, AND KEY BINDINGS

As described in the previous section, you use resources to specify mwm's appearance and behavior. You can also control how mwm's features are made available to you.

mwm has a large set of *window manager functions*. You use a function by *binding* it to a menu item, a mouse button, or a key, and then selecting that menu item or pressing that button or key. The `.mwmrc` file controls these bindings.

The .mwmrc File

The file `.mwmrc` in your home directory (or the file specified by the `configFile` resource, if different) specifies mwm's menus, button bindings, and key bindings. It consists of a series of named *binding sets*; each binding set consists of a series of *bindings*. A binding associates, or "binds," a window manager function to a menu item, mouse button, or key. You can have many binding sets in a `.mwmrc` file; the resources `windowMenu`, `buttonBindings`, and `keyBindings` determine which binding sets are currently in use.

If you don't have a `.mwmrc` file, your menus, button bindings, and key bindings come from the file `/usr/X/lib/system.mwmrc`. If you'd like to start with this default file and modify it to your taste instead of starting from scratch, use the command

```
$ cp /usr/X/lib/system.mwmrc $HOME/.mwmrc
```

and then edit the resulting `.mwmrc` file in your home directory.

Blank lines and lines beginning with an exclamation point (`!`) in the `.mwmrc` file are ignored. Any text between a pound sign (#) and the end of the line is also ignored. You can remove the special meaning of the exclamation point or pound sign by preceding it with a backslash (\).

You must restart mwm for changes in the `.mwmrc` file to take effect.

MWM Functions

The following pages describe all of mwm's window manager functions, in alphabetical order.

The *context* of a function (the window or icon affected by the function) is determined by the menu, button, or key that invokes it. The context of a menu or button is the window or icon containing the mouse pointer when the menu is popped up or the button is pressed, and the context of a key is the window or icon with the keyboard focus when the key is pressed. (The window containing the mouse pointer and the window with the keyboard focus can be different if the `keyboardFocusPolicy` resource is set to `explicit`.)

Some functions are not meaningful in certain contexts. If a function appears on a menu in a context in which it is not meaningful, it is grayed out and cannot be selected. If it is bound to a button or key in a context in which it is not meaningful, nothing happens when the button or key is pressed.

`f.beep`	Sounds a beep.
`f.circle_down`	Lowers the frontmost window or icon (sends it behind all other windows). Can be followed by the word `icon` to make it affect only icons, or by `window` to make it affect only windows.
`f.circle_up`	Raises the backmost window or icon (brings it in front of all other windows). Can be followed by the word `icon` to make it affect only icons, or by `window` to make it affect only windows.
`f.exec "`*command*`"`	Executes the given *command*, using the shell specified by `$SHELL` (or `/bin/sh` if `$SHELL` is not set). `f.exec` can be abbreviated by an exclamation point (`!`).
`f.focus_color`	Sets the *colormap focus* (see the `colormapFocusPolicy` resource) to the window in which it is executed. (Only meaningful if `colormapFocusPolicy` is `explicit`.)
`f.focus_key`	Sets the keyboard focus to the window in which it is executed.
`f.kill`	Closes the window in which it is executed. If the client has requested it, mwm notifies the client that the window should be deleted; otherwise, it simply terminates the window. (Not meaningful in the root window.)
`f.lower`	Lowers the window in which it is executed (sends it behind all other windows). If followed by *–client*, where *client* is a client name or class, lowers all windows with that name or class instead of the window in which it is executed.

`f.maximize` Maximizes the window in which it is executed (makes it fill the screen). (Not meaningful in a maximized window or the root window.)

`f.menu` *menu_name* Pops up the named menu. If this function is bound to a menu item, the named menu becomes a cascading (pull-right) menu from the current menu item. If bound to a button or key, the menu appears at the current cursor location when the button or key is pressed.

`f.minimize` Minimizes the window in which it is executed (turns it into an icon). (Not meaningful in an icon or the root window.)

`f.move` Moves the window in which it is executed. (Not meaningful in the root window.)

`f.next_cmap` Takes the next colormap in the colormap list of the window with the colormap focus and installs it as the server's colormap. (Only meaningful if the window with the colormap focus has more than one colormap,)

`f.next_key` Shifts the keyboard focus to the next window or icon in the stacking order. Can be followed by the word `icon` to make it step through only icons, by `window` to make it step through only windows, or by transient to make it step through only transient windows. (Only meaningful if `keyboardFocusPolicy` is `explicit`.)

`f.nop` Does nothing.

`f.normalize` Normalizes the window or icon in which it is executed (returns it to its normal size, shape, and position). (Not meaningful in a normalized window or the root window.)

`f.pack_icons` Arranges the icons in the window in which it is executed according to the value of the `iconPlacement` resource. (Icons that have been moved by the user are "packed" back to their standard positions.) (Only meaningful in the root window or icon box, depending on the value of the `useIconBox` resource.)

`f.pass_keys` Enables or disables processing of key bindings. When processing is enabled, keys that are bound in the binding set specified by the `keyBindings` resource invoke the specified window manager function. When processing is disabled, those keystrokes are passed to the window with the keyboard focus in-

stead of invoking the window manager function (this is useful for typing keys such as <Alt-space> in windows). Each time you invoke `f.pass_keys`, processing toggles between enabled and disabled.

`f.post_wmenu` Pops up the window menu (the menu specified by the `windowMenu` resource). If this function is bound to a button, the menu appears at the current cursor location when the button is pressed. If it is bound to a key, the menu appears in the upper left corner of the window with the keyboard focus when the key is pressed. (This function cannot be bound to a menu item.)

`f.prev_cmap` Takes the previous colormap in the colormap list of the window with the colormap focus and installs it as the server's colormap. (Only meaningful if the window with the colormap focus has more than one colormap,)

`f.prev_key` Shifts the keyboard focus to the previous window or icon in the stacking order. Can be followed by the word `icon` to make it step through only icons, by `window` to make it step through only windows, or by transient to make it step through only transient windows. (Only meaningful if `keyboardFocusPolicy` is `explicit`.)

`f.quit_mwm` Terminates `mwm`. If `mwm` is the "startup client," this also terminates the X Window System server. Otherwise, the server and any other clients keep running. (Only meaningful in the root window.)

`f.raise` Raises the window in which it is executed (brings it in front of all other windows). If followed by *–client*, where *client* is a client name or class, raises all windows with that name or class instead of the window in which it is executed.

`f.raise_lower` If the window in which this function is executed is partially or completely obscured by other windows, this function raises it. If the window is not obscured, this function lowers it. (Not meaningful in the root window.)

`f.refresh` Redraws all windows.

`f.refresh_win` Redraws the window in which it is executed. (Not meaningful in an icon or the root window.)

`f.resize` Resizes the window in which it is executed. (Not meaningful in an icon or the root window.)

`f.restart`	Restarts `mwm`, causing any changes you have made to resources or the `.mwmrc` file to take effect. (Only meaningful in the root window.)
`f.send_msg` *msg*	Sends the given message to the window in which it is executed. This function sends a `_MOTIF_WM_MESSAGES` message of type *msg* to the client, but only if the specified message number is included in the client's `_MOTIF_WM_MESSAGES` property. (Not meaningful in the root window.)
`f.separator`	Places a separator (horizontal line) in a menu. The label of the menu item to which this function is bound is ignored. (This function can only be used in a menu.)
`f.set_behavior`	Toggles `mwm` between the default Motif behavior and the behavior specified by the `.Xdefaults` and `.mwmrc` files.
`f.title`	Places a title (text that cannot be selected) in a menu. (This function can only be used in a menu.)

The following sections tell you how to bind these functions to menus, buttons, and keys.

Menu Bindings

The syntax of a menu binding set is:

```
Menu name
{
    label   [_character]   [key]   function
    label   [_character]   [key]   function
                .
                .
                .
    label   [_character]   [key]   function
}
```

The specified window manager *function* is executed when the *label* is selected from the menu, when the optional *character* key is pressed while the menu is popped up, or any time the optional *key* is pressed in the menu's context. (The *context* of a menu is determined by the button or key that pops the menu up, as described in the following sections.) Each binding must appear on one line. Any amount of white space can appear before the label and between the label, character, key, and function.

The *label* of a menu item can be either text or a bitmap. A text label may be enclosed in double quotes (""); it *must* be quoted if it contains a space or other special

character. A bitmap label is specified by `@`*pathname*, where *pathname* is the pathname of a bitmap file. (You can use the `bitmap` command to create a bitmap file.)

The optional *_character* specifies which character in a text label is underlined as a keyboard shortcut (pressing the key corresponding to the underlined character while a menu is popped up selects the appropriate item). The case of the *character* must match the character to be underlined in the label (but you must *not* hold down **<Shift>** or any other modifier key when using the keyboard shortcut).

The optional *key* specifies a keyboard shortcut for the menu item, in the following format:

[modifiers]`<Key>`*key_name*

The *key_name* is one of the key names from the file `/usr/X/include/X11/keysymdef.h`, with the `XK_` part removed. The optional set of *modifiers* is a space-separated list of modifier names, chosen from the following list, that specifies the modifier keys that must be held down while pressing the named key to invoke the menu item:

`Ctrl`	The **<Ctrl>** key
`Shift`	The **<Shift>** key
`Alt`	The **<Alt>** or **<Meta>** key
`Meta`	The **<Alt>** or **<Meta>** key
`Lock`	The **<Caps Lock>** key
`Mod1`	Modifier key #1 (**<Alt>** on a PC-compatible keyboard)
`Mod2`	Modifier key #2 (**<Num Lock>** on a PC-compatible keyboard)
`Mod3`	Modifier key #3 (not mapped on a PC-compatible keyboard)
`Mod4`	Modifier key #4 (not mapped on a PC-compatible keyboard)
`Mod5`	Modifier key #5 (not mapped on a PC-compatible keyboard)

No other modifier keys may be held down at the same time. (For example, if "`Alt<Key>x`" is specified, pressing **<Shift-Alt-x>** does *not* invoke the function.)

For example, here's the binding set for the standard window menu:

```
Menu DefaultWindowMenu
{
    "Restore"    _R  Alt<Key>F5    f.normalize
    "Move"       _M  Alt<Key>F7    f.move
    "Size"       _S  Alt<Key>F8    f.resize
    "Minimize"   _n  Alt<Key>F9    f.minimize
    "Maximize"   _x  Alt<Key>F10   f.maximize
    "Lower"      _L  Alt<Key>F3    f.lower
    no-label                       f.separator
    "Close"      _C  Alt<Key>F4    f.kill
}
```

Compare this binding set to the actual window menu to see how it works. For example, you can minimize a window by choosing "Minimize" from the window menu, by popping up the window menu and pressing the **<n>** key, or by pressing **<Alt-F9>** (with or without the menu popped up).

Note that the case of the *_character* matches the case of the underlined character in the label (compare "Size" to "Minimize").

Button Bindings

The syntax of a button binding set is:

```
Buttons name
{
    button_event   context   function
    button_event   context   function
                .
                .
                .
    button_event   context   function
}
```

The specified window manager *function* is executed when the *button_event* occurs while the pointer is in the specified *context*. Each binding must appear on one line. Any amount of white space can appear before the button event and between the button event, context, and function.

The *button_event* specifies a particular action with a particular button, in the following format:

[modifiers]<Btn*nAction*>

Where the Btn*nAction* is a single word that describes an action occurring on a particular mouse button. It consists of the letters Btn, followed by a single digit (*n*) from 1 (indicating the left mouse button) to 5 (indicating the rightmost mouse button on a five-button mouse), followed by one of the following *action* words:

Down	Pressing the button.
Up	Releasing the button.
Click	Pressing and releasing the button.
Click2	Pressing and releasing the button twice within the time specified by the doubleClickTime resource.

For example, <Btn1Down> specifies pressing and holding the left mouse button; <Btn2Click2> specifies double-clicking the middle mouse button.

The optional set of *modifiers* is the same as for a key: one or more words from the set Ctrl, Shift, Alt, Meta, Lock, Mod1, Mod2, Mod3, Mod4, and Mod5, separated from each other by spaces, specifying the modifier keys that must be held down while performing the *button_action* to invoke the function. For example, Alt<Btn1Down> specifies pressing and holding the left mouse button while holding down the **<Alt>** key; Shift Ctrl<Btn2Click2> specifies double-clicking the middle mouse button while holding down the **<Shift>** and **<Ctrl>** keys.

The *context* for a button binding is a series of words chosen from the following list:

root	The root (background) window.
icon	Anywhere in an icon.
window	Anywhere in a window or frame.
frame	Anywhere in a window frame.
title	In the title area of a window frame.
border	In the border area of a window frame.
app	In the content area of a window (within the frame).

If the context includes more than one word, they are separated from each other by vertical bars (|), indicating a logical OR. The function can be invoked by performing the button event in any of the named contexts. *No* white space may appear within a context specification.

For example, here's the standard button binding set:

```
Buttons PointerButtonBindings
{
    <Btn1Down>          frame|icon      f.raise
    <Btn2Down>          frame|icon      f.post_wmenu
    <Btn3Down>          frame|icon      f.lower
    <Btn1Down>          root            f.menu  RootMenu
    Meta<Btn1Down>      window|icon     f.raise
    Meta<Btn2Down>      window|icon     f.resize
    Meta<Btn3Down>      window|icon     f.move
}
```

With these standard button bindings:

- Pressing a mouse button while the pointer is in an icon or a window frame has the following effect:
  - The left button brings the window or icon to the front of the window stack.
  - The middle button pops up the window menu.
  - The right button sends the window or icon to the back of the window stack.
- Pressing the left mouse button in the root window pops up the root menu.
- Pressing a mouse button with the **<Alt>** key held down while the pointer is in an icon or *anywhere* in a window has the following effect:
  - **<Alt>**-left button brings the window or icon to the front of the window stack.
  - **<Alt>**-middle button resizes the window.
  - **<Alt>**-right button moves the window.

Note: If the focus policy is set to explicit by undefining the symbol `MWM_POINTER` in the file `/usr/X/lib/app-defaults/Mwm.rdb`, **<Alt>**-left button lowers the window instead of raising it.

Key Bindings

The syntax of a key binding set is:

```
Keys name
{
    key   context   function
    key   context   function
                .
                .
                .
    key   context   function
}
```

The specified window manager *function* is executed when the *key* is pressed while the pointer is in the specified *context*. Each binding must appear on one line. Any amount of white space can appear before the button event and between the button event, context, and function.

The *key* specification for a key binding is the same as the optional keyboard shortcut specification for a menu item: it has the format *[modifiers]*<Key>*key_name*, where the *key_name* comes from `/usr/X/include/X11/keysymdef.h` (with the `XK_` part removed) and the optional set of *modifiers* is one or more words from the set `Ctrl`, `Shift`, `Alt`, `Meta`, `Lock`, `Mod1`, `Mod2`, `Mod3`, `Mod4`, and `Mod5`, separated from each other by spaces.

The *context* for a key binding is the same as for a button binding: one or more words from the set `root`, `icon`, `window`, `frame`, `title`, `border`, and `app`, separated from each other by vertical bars (`|`).

For example, here's part of the standard key binding set:

```
Keys DefaultKeyBindings
{
   Shift<Key>Escape          icon|window
f.post_wmenu
   Meta<Key>space            icon|window
f.post_wmenu
   Meta<Key>Tab              root|icon|window    f.next_key
   Meta Shift<Key>Tab        root|icon|window    f.prev_key
   Meta<Key>Escape           root|icon|window    f.next_key
   Meta Shift<Key>Escape     root|icon|window    f.prev_key
}
```

With these key bindings:

- Pressing **<Shift-Esc>** or **<Alt-space>** while the pointer is in an icon or window pops up the window menu.
- Pressing **<Alt-Tab>** or **<Alt-Esc>** while the pointer is in the root window, an icon, or a window shifts the keyboard focus to the next window or icon in the stacking order. (Has no effect if the value of the `keyboardFocusPolicy` resource is not `explicit`.)
- Similarly, pressing **<Alt-Shift-Tab>** or **<Alt-Shift-Esc>** while the pointer is in the root window, an icon, or a window shifts the keyboard focus to the previous window or icon in the stacking order.

Example: Adding a Menu

Key bindings, button bindings, menus, and resources are all interrelated. Here's a small sample set of bindings that shows how to add a menu of your own.

Suppose you wanted to have a "remote logins" menu that pops up when you press the right mouse button while the pointer is in the root window. Here's how:

1. Add the following line to the file `.Xdefaults` in your home directory:

   ```
   Mwm*buttonBindings: MyButtonBindings
   ```

   This changes `mwm`'s button bindings from the standard bindings to the binding set `MyButtonBindings` in the `.mwmrc` file.

2. Copy the standard `.mwmrc` file to your home directory with the command

   ```
   $ cp /usr/X/lib/system.mwmrc $HOME/.mwmrc
   ```

3. Add the following lines to the end of your `.mwmrc` file:

   ```
   Buttons MyButtonBindings
   {
       <Btn1Down>          frame|icon      f.raise
       <Btn2Down>          frame|icon      f.post_wmenu
       <Btn3Down>          frame|icon      f.lower
       <Btn1Down>          root            f.menu  RootMenu
       <Btn3Down>          root            f.menu  MyMenu
       Meta<Btn1Down>      window|icon     f.raise
       Meta<Btn2Down>      window|icon     f.resize
       Meta<Btn3Down>      window|icon     f.move
   }
   ```

This button binding set is the same as the standard button bindings (PointerButtonBindings) with the addition of the "<Btn3Down> ... MyMenu" line. It adds one binding: pressing the right mouse button in the root context pops up the menu MyMenu, which is defined elsewhere in the .mwmrc file.

4. Add the following additional lines to the end of your .mwmrc file:

```
Menu MyMenu
{
  "Remote Logins"  f.title
  "venus"  f.exec "xterm -T venus -e rlogin venus &"
  "mars"   f.exec "xterm -T mars  -e rlogin mars &"
  "pluto"  f.exec "xterm -T pluto -e rlogin pluto &"
}
```

This menu binding set defines a menu titled "Remote Logins." It has three entries, each of which creates an xterm window. Each xterm performs an rlogin to the specified system and has its title set to the system name. (The rlogin command is part of the optional TCP/IP software package. See the rlogin and xterm manpages for more information on each.)

5. Finally, restart mwm (with the "Restart..." item on the root menu) to make your changes take effect.

   If you use xrdb, you must also use the command

   ```
   $ xrdb -merge $HOME/.Xdefaults
   ```

   before restarting mwm, to load the new value for the buttonBindings resource into the server's resource database.

If you try this and get unexpected results or no results, type the following command in any terminal window:

```
$ tail /usr/X/lib/xdm/xdm-errors
```

This will show you the last few error messages generated by the window system. If the messages in this file are not informative, contact Intel technical support.